A Comparison of Techniques for Scheduling Fleets of Earth-Observing Satellites

Al Globus
CSC
NASA Ames

James Crawford
RIACS
NASA Ames

Jason Lohn
NASA Ames

Anna Pryor
NASA Ames

Abstract

We have run experiments comparing fourteen techniques on a realistically-sized model of the Earth-Observing Satellite (EOS) scheduling problem. This problem requires taking as many high-priority observations as possible while satisfying complex constraints. We compared iterated sampling (ISAMP - basically random search), heuristic-based stochastic search (HBSS) with contention heuristics and multiple variants of the genetic algorithm, simulated annealing, squeaky wheel optimization and stochastic hill climbing on a three satellite, one week, 6,000+ observation scheduling problem. Simulated annealing with 'temperature dependent' random swap mutation operators was the clear winner. Random mutation operators outperformed squeaky mutation operators. HBSS with contention heuristics was hundreds of times slower than all other techniques and produced much worse schedules than all techniques except ISAMP. For this problem, at least among the techniques tested, simple, fast and stupid with learning significantly outperforms complex, slow and smart without learning.

Introduction

A growing fleet of scientific, military, and commercial Earth observing satellites (EOS) circles the globe. Most of these satellites are within about 700 km of the surface. Observations of any particular location can only be made only when the satellite is overhead. A single orbit takes approximately 100 minutes and, since the Earth turns underneath the satellite, any particular point on the surface is only occasionally, though predictably, visible. Although there are approximately 60 EOS satellites in orbit today, image collection is nearly always scheduled separately for each satellite with manual coordination, if any. Some studies (Globus *et al.* 2002) (Rao, Soma, & Padmashree 1998) have suggested that automatic coordination of multiple satellites can be beneficial, but the best scheduling techniques to use is not clear.

This study compares fourteen EOS scheduling techniques on a realistically-sized model problem. In particular, we compare simulated annealing, hill climbing, the genetic algorithm, squeaky wheel optimization, iterated sampling (ISAMP) and heuristic-based stochastic search (HBSS) (Bresina 1996) with contention heuristics (Frank *et al.* 2002). In the next section we describe the scheduling problem and our model. A description of the scheduling techniques follows. The nature and results of our computational experiments are then presented along with analysis. Space limitations preclude a review of previous work, but (Globus *et al.* 2002) and (Globus *et al.* 2003) provide such a discussion.

EOS Scheduling Problem

In this section we first describe the EOS scheduling problem as perceived by satellite operators and developers. Then we describe the model of the problem used in this experiment.

EOS scheduling attempts to take as many high-priority observations as possible within a fixed period of time on a fixed set of satellite-born sensors. For example, the Landsat 7 satellite scheduler is considered to have done a good job if 250 observations are made each day. There are generally far more than 250 observation requests. EOS scheduling is complicated by a number of important constraints. Potin (Potin 1998) lists some of these constraints as:

1. Revisit limitations. A target must be within sight of the satellite; and EOS satellites travel in fixed orbits. These orbits pass over any particular place on Earth at limited times so there are only a few observation windows (and sometimes none) for a given target.
2. Time required to take each image. Most Earth observing satellites take a one dimensional image and use the spacecraft's orbital motion to sweep out the area to be imaged. For example, a Landsat image requires 24 seconds of orbital motion.
3. Limited on-board data storage. Images are typically stored on a solid state recorder (SSR) until they can be sent to the ground.
4. Ground station availability (SSR dumps). The data in the SSR is sent to the ground when the satellite passes over a ground station. Ground station windows are limited as with any other target.
5. Transition time between look angles (slewing). Some instruments are mounted on motors that can point

side-to-side (cross-track).

6. Power and thermal control.
7. Coordination of multiple satellites.
8. Cloud cover. Some sensors cannot see through clouds. Not only do clouds cover much of the Earth at any given time, but some locations are nearly always cloudy.
9. Stereo pair acquisition or multiple observations of the same target by different sensors or the same sensor at different times.

For further details of the EOS scheduling problem see (Frank *et al.* 2002) and (Sherwood *et al.* 1998).

Our model problem implements all these constraints except the last two. The model problem consists of three satellites in Sun-synchronous orbits (orbits in which the equator is crossed a the same local time each orbit) for one week. The satellites are spaced ten minutes apart. Each satellite carries one sensor mounted on a cross-track slewable motor that can point up to 24 degrees to either side of nadir (nadir is straight down) and turns one degree in two seconds. Each satellite has an SSR capable of storing 50 arbitrary units.

We model our power and thermal constraints using so called duty cycle constraints, the approach taken by NASA's Landsat 7 satellite. A duty cycle constraint requires that the sensor not be turned on for longer that a maximum time within any interval of a certain length. This insures conformance with power, thermal, and other physical constraints on the spacecraft. Our model problem uses the Landsat 7 duty cycles. Specifically, a sensor may not be used for more than:

1. 34 minutes in any 100 minute period,
2. 52 minutes in any 200 minute period, or
3. 131 minutes in any 600 minute period.

There is one ground station in Alaska. Whenever a satellite comes within sight of the ground station it is assumed to completely empty it‘s SSR, which is then available for additional observation storage. There are approximately 75 SSR dumps per spacecraft during the week. Since some orbits are over oceans and all targets are on land, some SSR dump opportunities are wasted on an empty SSR.

6300 observation targets were randomly generated on land. Of these, 6114 were observable by at least one satellite during the one week scheduling period. The targets are assumed to be at the center of a rectangle that requires 24 seconds of satellite motion to image. Each observation requires one, three, or five arbitrary storage units (evenly distributed) on the SSR. Each observation was assigned a priority from one to six evenly spaced in 0.1 increments. Each observation has 2-24 windows, times when a satellite is within view of the observation‘s target. Orbits and windows were determined by the free version of the Analytical Graphics Inc.'s Satellite Tool Kit, also known as the STK (see www.stk.com).

The fitness (quality) of each schedule is determined by a weighted sum (smaller numbers indicate better fitness):

$$F = w_p \sum_{O_u} P_o + w_s S + w_a A \qquad (1)$$

where F is the fitness, O_u is the set of unscheduled observation, P_o is the priority of an observation, S is the total time spent slewing, A is the sum of the off-nadir pointing angle for all scheduled observations, w stands for weight, $w_p = 1$, $w_s = 0.01$, and $w_a = 0.00137$. Note that the weights favor the priority of unscheduled observations over pointing and slewing time objectives.

Scheduling Techniques

This study compares fourteen search techniques applied to the EOS scheduling problem. The simplest techniques were simulated annealing, hill climbing, two variants of the genetic algorithm, and ISAMP (essentially random search) taking random steps. By using a more intelligent mutation operator, these algorithms become variants of squeaky wheel optimization (Joslin & Clements 1999). Finally, we examined HBSS (Bresina 1996) with contention heuristics (Frank *et al.* 2002) where a great deal of processing is devoted to determining the order in which observations are placed in schedule timelines.

We represent a schedule as a permutation or ordering (the genotype) of the observations. A simple, deterministic, one-observation scheduler assigns resources to observations in the order indicated by the permutation (except for HBSS). This produces a timeline (the phenotype) with all of the scheduled observations, the time they were taken, and the resources (SSR, sensor, pointing angle) used. The one-observation scheduler assigns times and resources to observations using earliest-first scheduling heuristics while maintaining consistency with sensor availability, onboard memory (SSR) and slewing constraints. If an observation cannot be scheduled without violating the current constraints (those created by scheduling observations from earlier in the permutation), the observation is left unscheduled.

Simple earliest-first scheduling starting at epoch (time = 0) had some problems. We discovered that the algorithm works better if 'earliest-first' starts at some random time rather than at epoch. If the observation cannot be scheduled before the end of time, the algorithm starts at epoch and continues searching for a constraint-free window until the observation is scheduled or the initial time is reached. The time each observation is scheduled (or, if unscheduled, what time 'earliest-first' search started) is stored along with the permutation, is preserved by mutation and crossover, and is used as the starting point for the one-observation scheduler operating on modified versions of the current permutation. The extra scheduling flexibility may explain why this approach works better than earliest-first starting at epoch.

Constraints are enforced by representing sensors, slew-motors and SSRs as timelines. Scheduling an observation causes timelines to take on appropriate values (i.e., in use for a sensor, slew motor setting, amount of SSR memory available) at different times.

The simplest algorithm tested was ISAMP, which is essentially a random search. With ISAMP, each schedule is generated from a random permutation with random start times for the one-observation scheduler.

The next class of algorithms tested were the 'evolutionary' search techniques, which we define here as those that start with random permutations and generate new permutations with mutation and/or crossover. Unlike ISAMP, these algorithms learn in the sense that they use past experience and gradually improve the schedules generated. The algorithms tested were:

1. Stochastic hill climbing.
2. Simulated annealing. The temperature starts at 100 (arbitrary units) and is multiplied by 0.92 every 1000 children (100,000 children are generated per run).
3. A steady-state tournament selection genetic algorithm with population size 100. The individual to replace is chosen by a tournament from the whole population where the least fit is replaced. Tournament size is always two.
4. A generational elitist genetic algorithm. The population size is 110 where the 10 best individuals are copied into the next generation. Parents are chosen by tournament (size = 2).

Each search technique was tested with three mutation operators:

1. Random swap. Two permutation locations are chosen at random and the observations are swapped, with 1-15 swaps (chosen at random) per mutation. Earlier experiments determined that allowing more than one swap improved scheduling (Globus *et al.* 2003). A single random swap is called order-based mutation (Syswerda & Palmucci 1991).
2. Temperature-dependent swap. Here the number of swaps (1-15) is still chosen at random but with a bias. Early in evolution a larger number of swaps tend to be used, and later in evolution fewer swaps are performed. This is analogous to the 'temperature' dependent behavior of simulated annealing. The choice of the number of swaps is determined by a weighted roulette wheel where the weights vary linearly as evolution proceeds starting at n and ending at $15 - n$ where n is the number of swaps. Earlier experiments tried fewer swaps early in evolution and more swaps later. This didn't work as well.
3. Squeaky shift. This mutation operator implements squeaky wheel optimization. The mutator shifts 1-15 (chosen randomly) 'deserving' observations earlier in the permutation. Early in the permutation an observation is more likely to be scheduled since fewer other observations will have been scheduled creating additional constraints. Each observation to shift forward is chosen by a tournament of size 50, 100, 200, or 300 (chosen at random each time). The observation is always chosen from the last half of the permutation. The position-to-shift-in-front-of is chosen by a tournament of the same size (each time) and is guarrenteed to be at a location at least half way to the front of the permutation (starting at the 'deserving' observation). The observation most deserving to move earlier in the permutation is determined by the following characteristics (in order):

(a) unscheduled rather than scheduled
(b) higher priority
(c) later in the permutation

The position-to-shift-in-front-of tournament looks for the opposite characteristics.

We have tested a number of other mutation operators but the ones tested in this experiment performed the best. See (Globus *et al.* 2003) for some of these data.

In the case of the genetic algorithms half of all children are created by mutation and the other half by crossover. The crossover operator is Syswerda and Palmucci's position-based crossover (Syswerda & Palmucci 1991). Roughly half of the permutation positions are chosen at random (50% probability per position). The observations in these positions are copied from the father to the same permutation location in the child. The remaining observations fill in the child‘s other permutation positions in the order they appear in the mother.

The final algorithm tested was HBSS with contention heuristics. HBSS does not use the permutation or the one-observation scheduler. The observations are still scheduled one at a time, but the next observation to schedule is chosen by a weighted roulette wheel. For a given observation, the window (time when a satellite is in view of the target) to use is chosen by another weighted roulette wheel. Observations and windows are assigned dynamic weights which depend on the observations that have been scheduled (or found to be unschedulable) and windows that do not violate any constraint except the duty cycle constraint. The duty cycle constraint is not considered by the contention heuristics for performance reasons. If the chosen window is found to violate a duty cycle constraint it is discarded and another window chosen.

The weight of an observation is a function of how difficult it is to allocate a sensor and space on the SSR. An observations‘s difficulty is the minimum contention of the observation‘s windows. The contention of each window is a function of the other observation's windows that have incompatible sensor use requirements and/or compete for the SSR. This is modulated by the need of each observation, which is a function of an observation's priority and number of it‘s windows that do not violate the current constraints. The weight of a window is it‘s contention. The weighted roulette wheel that chooses the next window prefers windows with the lower weight. Formally:

$$\begin{aligned} D_o &= w_p P + w_s W_{mc}(C_s) + w_{ssr} W_{mc}(C_{ssr}) && (2) \\ C_s &= \sum_{O_s} N_s && (3) \\ C_{ssr} &= \sum_{O_{ssr}} N_{ssr} && (4) \\ N_s &= P/|W_a| && (5) \\ N_{ssr} &= SSR_a P/|W_a| && (6) \end{aligned}$$

where D_o is an observation's difficulty, P is an observation's priority, W_{mc} is the observation's window with the minimum contention over all of an observation's windows that are believed not to violate current constraints, C_s is a window's sensor contention, C_{ssr} is a window's SSR contention, w stands for weight, $w_p = 1$, $w_s = 1$, $w_{ssr} = 1$, O_s is the set of observations that cannot use a window's sensor if the window is scheduled due to sensor use or slewing constraints, N_s is an observation's sensor need, O_{ssr} is the set of observations that use the SSR between the same two SSR dumps and are not believed to violate current constraints, N_{ssr} is an observation's SSR need, SSR_a is the amount of memory currently available in the SSR, and W_a is the set of an observation's windows that are not believed to violate currant constraints; remember that the duty-cycle constraints are not considered in this assessment.

All of these values, except the priority and weights, are updated every time an observation is scheduled or found to violate the constraints, including the duty cycle constraint. This requires a great deal of complex memory and CPU intensive code.

Note that the weights are simply set to one. There may be a more optimal setting of these weights but finding betters weights involves running the scheduler a great deal which is computationally expensive. With a simpler problem, a search for better weights did not improve schedule quality.

HBSS with contention heuristics is hundreds of times slower than the other techniques on this problem, and thousands of times slower on smaller problems with much slower slewing motors (more windows are incompatible with each other). The contention heuristics require large, complex data structures to incrementally update the need, contention, difficulty and weight of each as-yet-unscheduled observation and it's windows. This requires substantial memory. For example, this model problem with no limits on the slew angle could not fit in 400 megabytes (there were more observation windows and more were incompatible with each other).

Experiment

To find the best algorithm for the model problem we compared a total of fourteen techniques. These were ISAMP, HBSS, and every combination of four search techniques – hill climbing, simulated annealing, steady state GA, and generational GA – with three mutation operators – 1-15 random swaps, 1-15 temperature dependent swaps, and 1-15 squeaky shifts. Except for HBSS, 32 jobs with identical parameters (except the random number seed) were run for each algorithm. Each job generated approximately 100,000 schedules (the GA runs generated slightly more). On one Athlon processor of our Linux cluster these jobs took 2-3 hours each.

By using our entire cluster for HBSS, we were able to run the equivalent of eight 100,000 schedule jobs. Generating only 10,000 HBSS schedules takes about 69 hours on the same Athlon processor, so its a couple hundred times slower than the other techniques. Because the HBSS data are different they are not included in tables and figures to avoid the impression the data are directly comparable. However, the HBSS results were so poor (barely better than ISAMP) that the difference in data size makes little difference to the conclusions.

Table 1 compares the mean fitness. Nearly all of the differences were statistically significant by both t-test and ks-test, with confidence levels usually far above 99%. We see that simulated annealing with temperature dependent swaps (SaTd) performs best with algorithms using simulated annealing, hill climbing, temperature dependent swaps, and random swaps clearly leading. Interestingly, although temperature dependent swaps won with simulated annealing and hill climbing, random swaps was superior for both genetic algorithms. The message seems to be modify one schedule, take random steps, and restrict steps more and more as evolution proceeds.

Simulated annealing and hill-climbing with the squeaky shift operator were next. These outperform the genetic algorithm regardless of mutation operator. Again, within the genetic algorithms the squeaky shift mutator performs the worst.

ISAMP, as one might expect for random search, performed the worst. However, the very best HBSS schedule (out of 800,000 generated) was only a little bit better than the ISAMP mean.

The small standard deviations for all techniques suggests that all runs for a given technique get about the same fitness. Thus, even if the fitness landscape is multi-modal all the minima must be about the same. Figure 1, which shows the breadth of each fitness distribution over 32 runs, confirms this view. For this reason, we suspect that this problem requires mostly exploitation, rather than exploration, which also explains the poor GA results. Evolutionary change is spread out over the GA populations rather than concentrated on a single individual as for simulated annealing and hill climbing.

The squeaky shift mutator's performance relative to random swaps suggests that it is smart in the wrong way. In preliminary experiments we also tried swapping, rather than shifting, observations and forcing observations to be swapped into certain parts of the permutation. The shift operator performed the best, but still not as well as the random swap mutator. See

(Globus *et al.* 2003) for some of this data. If random outperforms intelligent, then clearly the intelligence is being applied in the wrong way. We do not understand the dynamics of permutation scheduling in any fundamental way, and we don't even know if the dynamics are fundamentally similar for different problems. Until a better understanding is reached, the random swap operators – with a decrease in the number of swaps as evolution proceeds – appear best.

Figures 2-4 show that the individual objectives in the weighted sum of equation 1 display much the same trend as the fitness. Simulated annealing and hill climbing with random swaps beats squeaky shifts and the genetic algorithm in almost every objective of the fitness function. However, notice that the range of average off-nadir pointing is very large suggesting that this measure made little difference, perhaps because the weight was too low.

Figure 5 compares the number of unscheduled observations, an objective not found in the fitness function. Notice that the squeaky mutators in simulated annealing and hill climbing are worse than the genetic algorithms with random swaps. This suggests that the squeaky mutators with simulated annealing and hill climbing do a better job of scheduling the high priority observations to make up for scheduling fewer observations.

It is difficult to say precisely why HBSS did so poorly compared to simulated annealing and the other techniques. Perhaps because the essence of the problem is scheduling the observations in the proper order. HBSS attempts to discover this order using the contention heuristics. Perhaps the heuristics are simply the wrong ones. However, perhaps the ability of simulated annealing and the other techniques to discover, and preserve, partial orders of observations to schedule is the essence of their superiority.

Summary

We compared fourteen different techniques for scheduling EOS fleets on a realistically-sized model problem. Simple techniques such as simulated annealing and hill climbing outperformed the genetic algorithm and HBSS with contention heuristics. Simple random swap mutation outperformed more 'intelligent' mutation. Reducing the number of random swaps as evolution proceeds also improves performance. The most 'intelligent' algorithm with no learning, HBSS with contention heuristics, barely outperformed random scheduling (ISAMP) in spite of requiring far more computing resources. Although we examined only one problem here, we have seen essentially the same results on other problems in this class. For some of this data see (Globus *et al.* 2003). Apparently, taking advantage of previous scheduling attempts, as simulated annealing, hill climbing, and the genetic algorithm does, has more value than large amounts of computation to choose just the right move to make. For this application and these techniques, fast and stupid with a little learning outperforms smart.

Acknowledgements

This work was funded by NASA's Computing, Information, & Communications Technology Program, Advanced Information Systems Technology Program (contract AIST-0042), and by the Intelligent Systems Program. Thanks to Alex Herz, Orbit Logic, Inc. and Jerald Arp, PhD., Manager of Technical Support, Space Imaging LLC for information regarding current EOS systems. Thanks also to Bonnie Klein for reviewing this paper and to Jennifer Dungan, Jeremy Frank, Robert Morris and David Smith for many helpful discussions. Finally, thanks to the developers of the excellent Colt open source libraries for high performance scientific and technical computing in Java (http://hoschek.home.cern.ch/hoschek/colt).

References

Bresina, J. 1996. Heuristic-biased stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. 2002. Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space 2002*.

Globus, A.; Crawford, J.; Lohn, J.; and Morris, R. 2002. Scheduling earth observing fleets using evolutionary algorithms: Problem description and approach. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2003. Scheduling earth observing satellites with evolutionary algorithms. In *Conference on Space Mission Challenges for Information Technology (SMC-IT)*.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Potin, P. 1998. End-to-end planning approach for earth observation mission exploitation. In *SpaceOps 1998*.

Rao, J. D.; Soma, P.; and Padmashree, G. S. 1998. Multi-satellite scheduling system for leo satellite operations. In *SpaceOps 1998*.

Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Using aspen to automate eo-1 activity planning. In *Proceedings of the 1998 IEEE Aerospace Conference*.

Syswerda, G., and Palmucci, J. 1991. The application of genetic algorithms to resource scheduling. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 502–508.

Search technique	mutation operator	abbreviation	mean fitness	fitness std. dev.
simulated annealing	temperature dependent swaps	SaTd	9205	20
hill climbing	temperature dependent swaps	HcTd	9310	21
simulated annealing	random swaps	SaSr	9311	19
hill climbing	random swaps	HcSr	9368	25
simulated annealing	squeaky swaps	SaSs	9489	19
hill climbing	squeaky swaps	HcSs	9507	24
generational GA	random swaps	GgSr	9700	38
steady state GA	random swaps	GsSr	9700	25
steady state GA	temperature dependent swaps	GsTd	9741	31
generational GA	temperature dependent swaps	GgTd	9834	24
generational GA	squeaky swaps	GgSs	9964	53
generational GA	squeaky swaps	GsSs	10010	46
ISAMP	random	ISAMP	10463	11

Table 1: Scheduling algorithms tested ordered by mean fitness. Smaller values indicate better fitness. HBSS is left out since processing time did not permit 32 HBSS runs of 100,000 schedules each. The best HBSS fitness in the equivalent of 8 runs was 10442, a little better than the ISAMP mean but worse than the worst run for all other techniques. The closest worst-run fitness value was 10120 (222 better than the HBSS best value) for the steady-state genetic algorithm with squeaky shifts (GsSs).

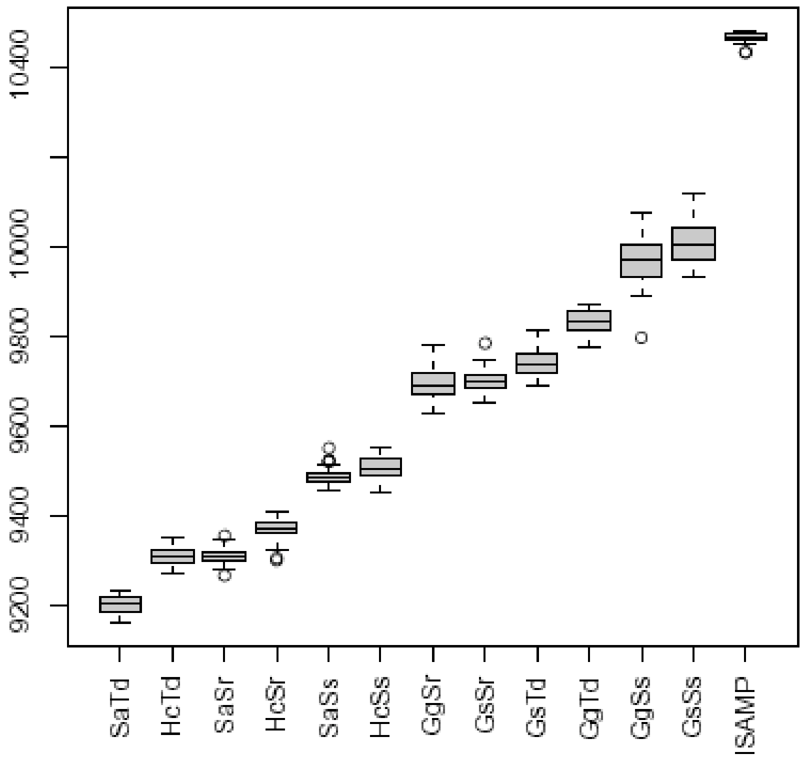

Figure 1: Comparing fitness (vertical axis) for 32 runs. The boxes indicate the second and third quartiles. The line inside the box is the median and the whiskers are the extent of the data. Outliers are represented by small circles. Smaller numbers indicate better fitness.

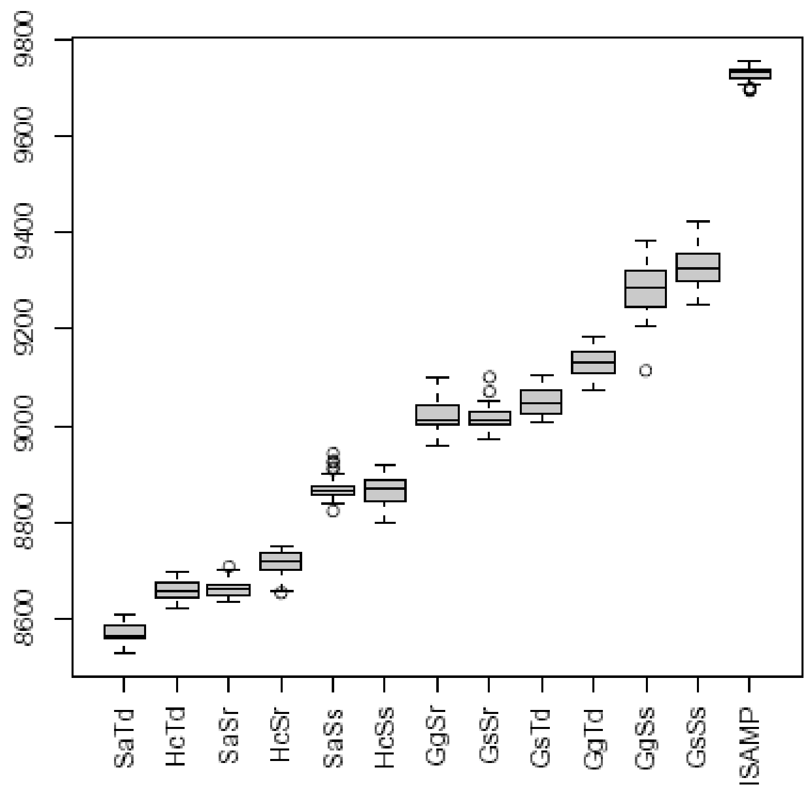

Figure 2: Sum of the priority of unscheduled observations ($\sum_{O_u} P_o$ from equation 1).

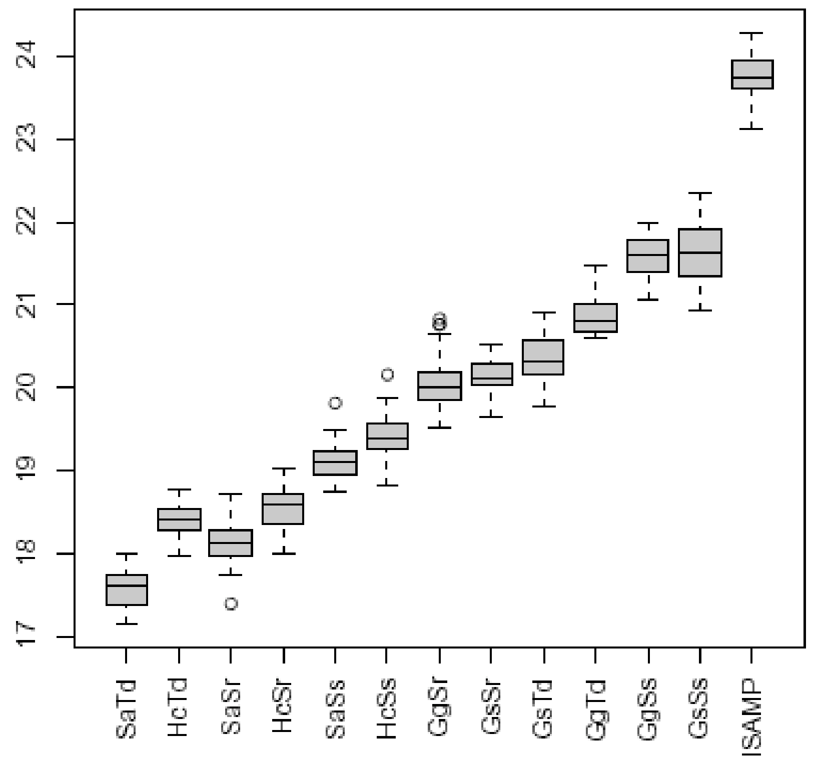

Figure 3: Mean slewing time needed for each scheduled observation (mean of S from equation 1).

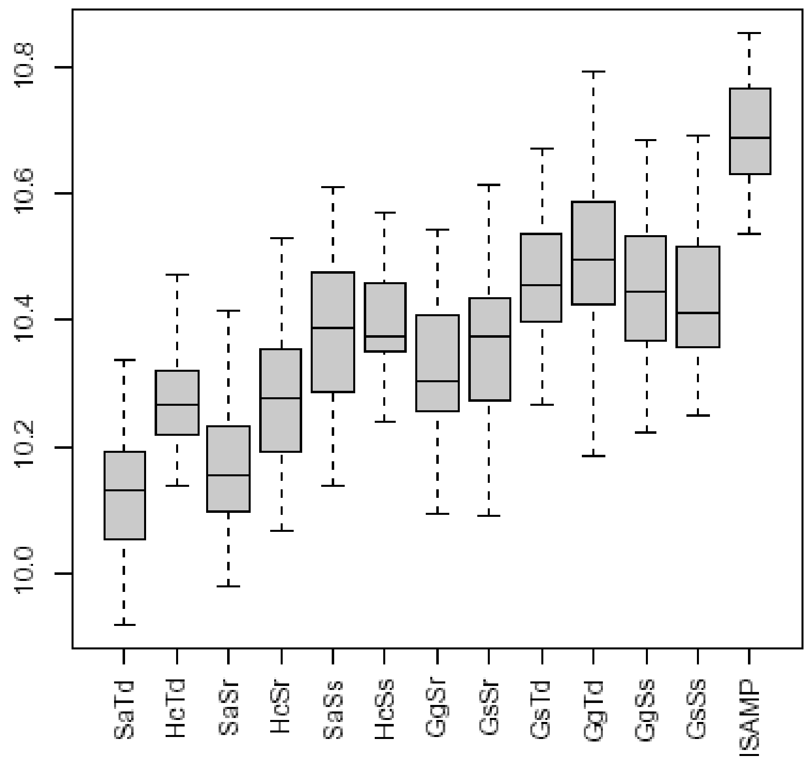

Figure 4: Mean off-nadir pointing angle needed for each scheduled observation (mean of A from equation 1).

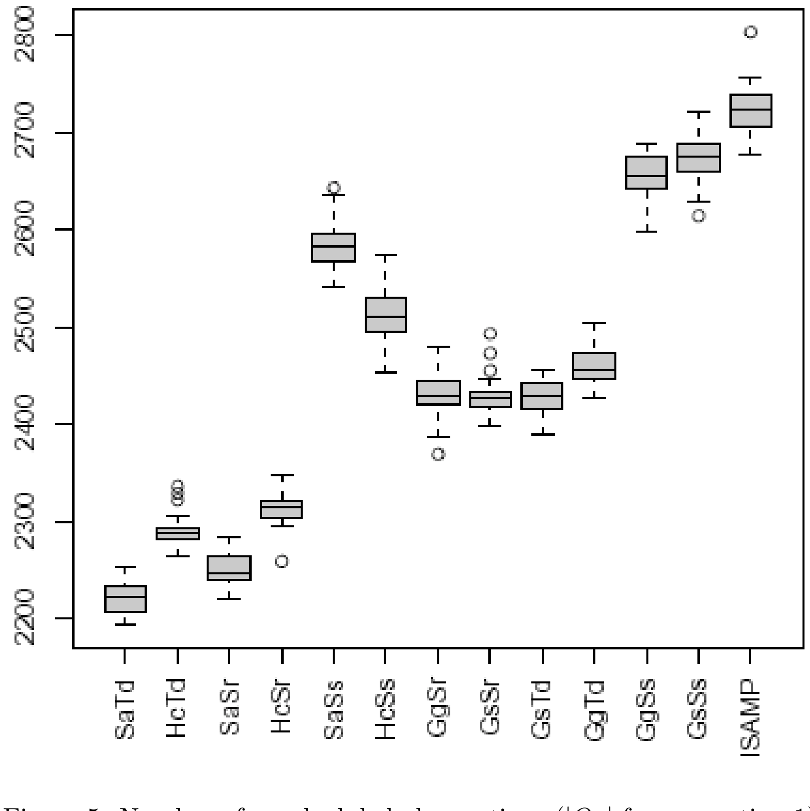

Figure 5: Number of unscheduled observations ($|O_u|$ from equation 1).